# Service Oriented Protocols for Human Computation

Daniel Schall

**Abstract** Human computation and crowdsourcing are increasingly gaining momentum. Many platforms already exist providing basic features for crowdsourcing different types of tasks to the Web. Service Oriented Architectures (SOA) provide the ideal technical framework to support interactions with both Human-Provided Services (HPS) and Software-Based Services (SBS). A unified service-oriented computing approach allows to combine the capabilities of humans and software services. Here we discuss the functional and non-functional requirements of service-oriented protocols for human computation. Human interactions in service-oriented systems need to be enabled in a different manner than interactions with software services. We describe the mapping of human interactions onto a service-oriented infrastructure.

## 1 Introduction

Human computation is increasingly becoming mainstream and commonplace in collaborative computing environments. This is partially due to the success of applications such as 'games with a purpose' [16] (a game wherein two people need to agree on a common set of keywords which they associate with images) and the success of commercially available crowdsourcing platforms such as Amazon Mechanical Turk (AMT) [2]. Human computation and crowdsourcing are often used interchangeably to address problems that computers cannot yet tackle on their own in an efficient manner [16]. In a similar spirit, crowdsourcing is commonly defined as *'the act of taking a job traditionally performed by a designated agent and outsourcing it to an undefined, generally large group of people in the form of an open call'* [7]. A recent taxonomy and survey [10] overviews existing literature in the context of human computation and crowdsourcing.

Daniel Schall
Siemens Corporate Technology, Vienna, Austria, e-mail: daniel.schall@siemens.com

Many crowdsourcing systems have recently emerged on the World Wide Web [6] including CrowdFlower [5], oDesk [9], ClickWorker [4], SmartSheet [14], and SpeechInk [15]. These platforms allow people to work on tasks such as transcription of spoken language into text, translation of text, tagging of images, and coding as well as integration of scripts and APIs. By nature, platforms on the World Wide Web are under constant flux and change. Also, human computation and crowdsourcing platforms are dynamic with people around the globe joining and leaving communities [8]. Thus, it is essential to account for the dynamics in the availability of a large-scale human workforce, changing skills of crowd workers, and changing requirements with regards to the underlying communication protocol.

Protocols for human computation and crowdsourcing have to account for these dynamic aspects by supporting

- adaptive (flexible) interactions that may span numerous human and software services as well as a range of devices,
- monitoring and logging mechanisms to observe the environment and ongoing interactions and
- a crowd worker discovery mechanism based on workers' capabilities, evolving skills and availability constraints.

In previous work we proposed service-oriented computing principles to address the challenges of human computation and crowdsourcing in large-scale environments. *Mixed service-oriented systems* [11] consist of Human-Provided Services (HPS) [13] and Software-Based Services (SBS) that can be composed to jointly solve crowdsourcing tasks [12]. The novelty of mixed service-oriented computing environments is the application of social principles to coordinate the execution of human tasks within open Web-based systems. Existing XML-based industry standards such as WS-HumanTask (WS-HT) [3] and Bpel4People (B4P) [1] can be integrated into mixed service-oriented systems to support the coordination of a set of distributed human tasks. *Non-functional requirements* play an essential role for the integration of these standards in open computing environments because of the inherent dynamic nature of Web-based systems. Our prior work [12] provides the basis for the following discussions on service-oriented protocols for human computation.

We start with an overview of the system context in Sect. 2 detailing the environment in which human computation is performed. In this work we discuss the basic functional and non-functional protocol requirements with regards to human computation (see Sect. 3). Based on the basic protocol requirements we describe the mapping of human interactions onto a service-oriented infrastructure in Sect. 4. This is detailed at a technical level by providing an actual XML-based description of a Human-Provided Services interface.

## 2 System Context Overview

A general system context overview is provided by Fig. 1. Figure 1 shows various actors and their roles and a set of architectural building blocks. The essential roles within a human computing or crowdsourcing environment are listed and described in the following.

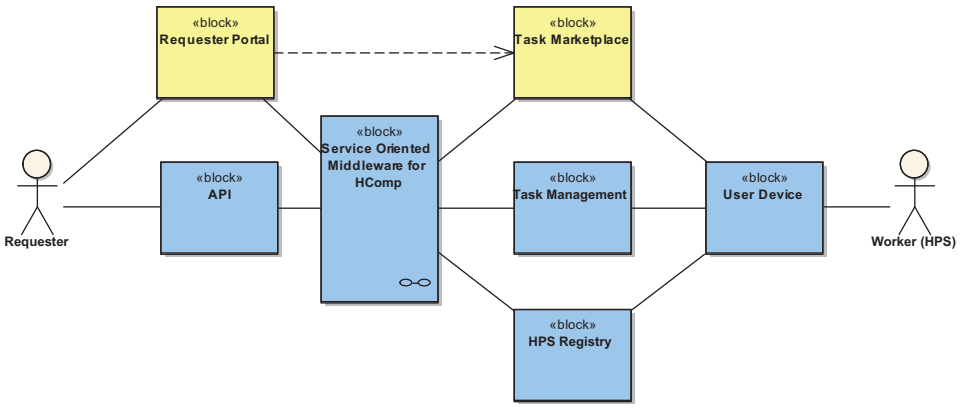


**Fig. 1** Main architectural building blocks.

- **Platform Provider**: the platform provider (not shown in Fig. 1) is responsible for providing and maintaining the crowdsourcing platform. The platform provider must also ensure that enough workers are available for task processing (e.g., by providing incentive mechanisms) and that requesters post enough tasks so that workers are able to obtain rewards.
- **Requester**: the requester sends human tasks to the platform. The requester may be a person who wishes to crowdsource a set of tasks or a software agent that needs to outsource certain steps in a computational process to a human. The requester typically pays a monetary reward to the workers as well as the platform provider.
- **Worker (HPS)**: the worker or Human-Provided Service (HPS) [13] either claims or receives a human task. The HPS concept enables the seamless integration of human capabilities into a service-oriented human computation or crowdsourcing environment [12]. An HPS can be discovered like an SBS. However, interactions with an HPS need to be performed in a manner suitable for a human. The result of the human task is returned by the HPS and delivered to the requester.

From a technical point of view, various building blocks are needed to realize a service-oriented platform for human computation. First, we briefly discuss two building blocks that common systems such as AMT implement to realize a marketplace for task-based crowdsourcing. However, market-based crowdsourcing is not focus of this work.

- **Requester Portal**: the requester has the ability to create new tasks and monitor the progress of human tasks using a Web-based portal.
- **Task Marketplace**: human tasks may be presented in the task marketplace. Workers (HPS) have the ability to discover and claim new tasks that are available through the marketplace.

The following building blocks can be found in a service-oriented human computation environment:

- **API**: The requester has the ability to create human tasks through the API programmatically. This allows to integrate human computation techniques into existing infrastructures.
- **Service Oriented Middleware for HComp**: The middleware implements features such as HPS discovery, asynchronous interaction support, monitoring and mining of interactions.
- **Task Management**: The task management provides a standardized interface to create and manipulate task instances. For that purpose, existing standards such as WS-HT [3] can be adopted that already define the various states of generic human tasks.
- **HPS Registry**: A Human-Provided Service can offer the capability to, for example, 'translate documents', perform document review', or 'provide help and support'. The HPS Registry helps to discover the demanded service by performing a lookup procedure.
- **User Device**: Workers have the ability to use their preferred device to interact with the system. This may demand the adaptation of the user interface.

## 3 Basic Protocol Requirements

In this section we discuss the most important functional and non-functional requirements of a service-oriented protocol for human computation.

**Functional requirements** include:

1. The protocol must support *asynchronous interactions* between requesters and workers (HPS). Humans operate at a different speed than software services and thus all interactions should be performed asynchronously. This is true for most interactions where human input is needed.
2. The protocol must support seamless *interactions and composition of human and software services*. A seamless infrastructure integrates the capabilities of people and software services to perform computation in a hybrid service-oriented system.
3. The protocol must support *well defined interfaces* that define the possible interactions. Well defined interfaces help to discover the appropriate service (HPS).

**Non-functional requirements** must be defined to provide assurance with regards to different qualities:

1. The requester must be able to explicitly state non-functional requirements in form of *service-level agreements (SLAs)*. SLAs must be stated so that the middleware platform is able to interpret and enforce the negotiated agreements.
2. All *interactions must be monitored* to provide historical information that can be used for later analysis. Indeed, monitored information is only used by the platform itself to perform, for example, ranking and selection of the most suitable HPS. Selecting skilled workers helps to improve or guarantee the quality of delivered task results.
3. *Interactions must support flexibility* to enable load balancing and delegation. Since human tasks may arrive in an unpredictable manner (bursts), HPSs must be able to perform task delegation to balance their workload. This should be assisted by the platform to help finding the appropriate delegation receiver.

The following section details the protocol and describes how the presented functional and non-functional requirements are satisfied by the protocol.

## 4 Service Oriented Protocol

We propose the application and extension of existing Web service standards for human computation and crowdsourcing. Here we present a concrete HPS interface example and discuss how the discussed requirements are addressed by the proposed protocol and Web service standards.

**Technical description.** Technical service interfaces are typically described by using the well-established Web Services Description Language (WSDL)[1]. WSDL interfaces help to discover and invoke services through a late binding mechanism. The very same description language can be used to describe an HPS.

Listing 1 shows the definition of a WSDL interface of a *translation service* (to translate a document from one language to another) that is used to interact with people in a service-oriented manner. Using WSDL as the interface description language brings the important advantage that the same standard is used to describe both HPS and software services (SBS). The WSDL in Listing 1 is automatically generated by the Service Oriented Middleware for HComp (see Fig. 1). It shows also the structure of the complex data type that is passed to the HPS to perform the actual task. Type information is used to automatically generate XML-based graphical user interfaces using forms technologies (see XForms[2]).

Notice, human task related information in managed by a separate task management service and is not depicted by the presented HPS interface. Task management can be implemented as a WS-HumanTask infrastructure (see [3, 12]). This allows for a clear separation of the generic task model (task states, transitions, etc.) and the actual application specific service (HPS) model. Thus, HPSs can be designed at any time and registered with the HPS registry (see Fig. 1).

[1] `http://www.w3.org/TR/wsdl`

[2] `http://www.w3.org/MarkUp/Forms/`

```
1  <wsdl:definitions name="TranslationService" ...>
2   <wsdl:types>
3    <xs:schema elementFormDefault="unqualified" tns="http://...">
4     <xs:element name="assignProcRequest"
5      type="tns:assignProcRequest" />
6     <xs:element name="getStatus" type="tns:getStatus" />
7     <xs:element name="getProcResult" type="tns:getProcResult" />
8     <!-- responses omitted-->
9     <xs:complexType name="desc">
10     <xs:element name="docTitle" type="xs:string" />
11     <xs:element name="docUri" type="xs:string" />
12     <xs:element name="length" type="xs:string" />
13     <xs:element name="language" type="xs:string" />
14     <xs:element name="translation" type="xs:string" />
15     <xs:element name="translationUri" type="xs:string" />
16     <xs:element name="mimeType" type="xs:string" />
17     <!-- further details omitted-->
18    </xs:complexType>
19    <!-- other types... -->
20   </xs:schema>
21  </wsdl:types>
22  <wsdl:message name="assignProcRequest">
23   <wsdl:part element="tns:assignProcRequest" name="params" />
24  </wsdl:message>
25  <!-- messages... -->
26  <wsdl:portType name="TS">
27   <wsdl:operation name="assignProcRequest">
28    <!-- in-/output... -->
29   </wsdl:operation>
30  </wsdl:portType>
31  <wsdl:binding name="TSSoapBinding" type="tns:TSService">
32   <soap:binding style="document" transport="http://schemas..."/>
33    <!-- operations... -->
34  </wsdl:binding>
35  <wsdl:service name="TSService">
36   <wsdl:port binding="tns:TSSoapBinding" name="TSPort">
37    <soap:address location="http://somehost:8080/..."/>
38   </wsdl:port>
39  </wsdl:service>
40 </wsdl:definitions>
```

**Listing 1** Interface description of human translation service: description is used to define complex data types and to support discovery of HPS.

The elements in Listing 1 are used for the following purpose:

- `docTitle`: defines the name of the document to be translated by a human.
- `docUri`: contains the location (e.g., link to a document repository) where the document can be downloaded from.
- `length`: the length (number of words) of the document.
- `language`: specifies the language in which the document is written.

- `translation`: the target language to be translated to (e.g., provide translation from German to English).
- `translationUri`: the location (repository) of the translated document. The person translating the document may provide an alternative location that can be requested via the Web service operation `getProcResult`.
- `mimeType`: states the acceptable format of the translated document (e.g., PDF, Word document, or plain text).

To support monitoring and logging, interactions are captured through XML message interceptors deployed within the service runtime environment. Messages are saved in a log database for analysis. An example interaction log is shown by Listing 2, which includes various message header extensions for message correlation and context-aware interaction analysis.

```
1  <soap:Envelope
2   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
3   xmlns:hps="http://www.danielschall.at/hps/">
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xmlns:wsa="http://schemas.xmlsoap.org/ws/.../addressing"
6   <soap:Header>
7    <hps:Timestamp value="2013-03-07T17:24:18"/>
8    <hps:TaskUri="http://.../HumanTask#42"/>
9    <wsa:MessageID>uuid</wsa:MessageID>
10   <wsa:From>http://.../Actor#Actor1</wsa:From>
11   <wsa:To>http://.../Actor#Actor2</wsa:To>
12   <wsa:ReplyTo>http://.../Actor#Actor3</wsa:ReplyTo>
13   <wsa:Action>http://.../Type/Translate</wsa:Action>
14  </soap:Header>
15  <soap:Body>
16   <hps:Request>
17    <!-- request omitted -->
18    <hps:keywords>document, translation</hps:keywords>
19   </hps:Request>
20  </soap:Body>
21 </soap:Envelope>
```

**Listing 2** HPS message log example: logs are used to perform analysis of interactions.

The purpose of the most important extensions is outlined in the following:

- `Timestamp` captures the actual creation of the message and is used to calculate temporal interaction metrics, such as average response time.
- `TaskUri` describes the context of interactions based on the task performed by the user. The `TaskUri` helps to correlate messages and task context.
- `MessageID` enables message correlation, i.e., to properly match requests and responses.
- `WS-Addressing` extensions, besides `MessageID`, are used to route requests through the collaborative (social) network. Routing is performed through delegation but can also be assisted by the middleware through a rule based system.

**Addressing functional and non-functional requirements.** The functional and non-functional requirements are satisfied as follows:

- Asynchronous interactions (see functional requirement 1) are supported through the operations `assignProcRequest` and `getProcResult`.
- Interactions and composition of human and software services (see functional requirement 2) are supported because the same technical standards and framework are used.
- Well defined interfaces (see functional requirement 3) are supported through the use of well defined XML-based WSDL interfaces.
- Service-level agreements (see non-functional requirement 1) are technically supported through the Web services stack. A detailed discussion is provided in [12].
- Interactions are monitored (see non-functional requirement 2) to provide mechanisms for temporal analysis, message and task correlation, and fine-grained expertise analysis.
- Interactions can be performed in a flexible manner (see non-functional requirement 3) through delegation patterns (at the technical level, WS-Addressing[3] mechanisms help to route messages).

## 5 Conclusions

In this chapter we have discussed service oriented protocols for human computation. Service oriented protocols help implementing applications for human computation such as the presented translation service and other possible applications such as GWAP. The main advantage of service oriented protocols is the potential integration of crowdsourcing into business environments that are based on Web services technologies and related BPM standards. Interface design of human-based services is an important issue. Using the Web service description language in combination with SOAP is an effort to standardize interface descriptions for human computation. These formal XML-based standards help defining domain concepts as data types in a rigorous manner. Other more Web-centric data formats such as JSON based data types may also be used to exchange task requests with human-based services. However, these formats currently lack a standardized approach for describing service interfaces. Web-centric data formats in the context of human computation will be analyzed in our future work.

## References

1. A. Agrawal et al. Ws-bpel extension for people (bpel4people), version 1.0., 2007.
2. Amazon. Online: mturk.com (last accessed 05-Mar-2013).

[3] `http://www.w3.org/Submission/ws-addressing/`

3. M. Amend et al. Web services human task (ws-humantask), version 1.0., 2007.
4. ClickWorker. Online: clickworker.com (last accessed 05-Mar-2013).
5. CrowdFlower. Online: crowdflower.com (last accessed 05-Mar-2013).
6. A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96, Apr. 2011.
7. J. Howe. The rise of crowdsourcing. *Wired*, 14(14):1–5, 2006.
8. P. G. Ipeirotis. Analyzing the amazon mechanical turk marketplace. *XRDS*, 17:16–21, Dec. 2010.
9. oDesk. Online: odesk.com (last accessed 05-Mar-2013).
10. A. J. Quinn and B. B. Bederson. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, CHI '11, pages 1403–1412, New York, NY, USA, 2011. ACM.
11. D. Schall. A human-centric runtime framework for mixed service-oriented systems. *Distributed and Parallel Databases*, 29(5-6):333–360, 2011.
12. D. Schall. *Service Oriented Crowdsourcing: Architecture, Protocols and Algorithms*. Springer Briefs in Computer Science. Springer New York, New York, NY, USA, 2012.
13. D. Schall et al. Unifying human and software services in web-scale collaborations. *IEEE Internet Computing*, 12(3):62–68, 2008.
14. SmartSheet. Online: smartsheet.com (last accessed 05-Mar-2013).
15. SpeechInk. Online: speechink.com (last accessed 05-Mar-2013).
16. L. von Ahn. Games with a purpose. *IEEE Computer*, 39(6):92–94, 2006.